Development (from prototype to widely used)

Deb Agarwal (LBNL) and Paul Iwanchuk (LANL)

- Testing, Tracking (results, bugs, dependencies)
- Continuous Integration, Agile
- Validation and Verification

- Software engineering best practices typically include a thorough regimen of testing, bug tracking, documentation and release. Software design practices such as agile development and continuous integration are widely employed in developing code. An HPC environment brings with it several unique aspects including that development of software for HPC systems is often concurrent with the maturation of the target system. HPC software includes the applications, the libraries, the operating system as well as software targeted to testing the software and hardware environment. Validation and verification play a central role along with regression testing, tracking and documenting results. Similarly, there is a concerted effort to assure key applications are "ready" for the new architecture.
- This session will focus on the life cycle development and maintenance practices that are key to successful deployment, and operation of these software components. We will follow software practices in the maturation of a typical HPC system from procurement through production to end of life. This session will address best practices at these stages, rather than addressing software components. Questions such as: How do you assure production readiness? What is your reliance on in-house development vs. vendor support? Is your custom environment helping or hindering end use? What is your ability to use other DOE institutional resources to complete your mission. What are the barriers? For DST's and updates? What role is fault tolerance and resilience playing in your future?

Session 2 Participants

Deb Agarwal	LBNL	Session Co-Chair
daagarwal@lbl.gov		
Paul Iwanchuk	LANL	Session Co-Chair
pniwanc@lanl.gov		

Name Organization email Breakout 2

Chris Atwood HPCMPO / CREATE	atwood@scalednumericalphysics.com
Shreyas Cholia NERSC/LBL	scholia@lbl.gov
Janet Lebens Cray Inc.	jml@cray.com
Bob Ciotti NASA Advanced Supercomputing (NAS) Division	Bob.Ciotti@nasa.gov
Kenneth Alvin Sandia Natl Labs	kfalvin@sandia.gov
Dong Ahn DEG/SDD/ICCD	ahn1@llnl.gov
Ann Gentile Sandia National Laboratories	gentile@sandia.gov
Steve Cotter ESNet/Lawrence Berkeley Laboratory	steve@es.net
Andrew Hanushevsky SLAC National Accelerator Lab	abh@stanford.edu
Jim Brandt Sandia National Laboratory	brandt@Sandia.gov

Name Organization email Breakout 2

Ewing Lusk Argonne National Laboratory	lusk@mcs.anl.gov
Lie-Quan Lee SLAC National Accelerator Lab	liequan@slac.stanford.edu
gina morello admin	gina.f.morello@nasa.gov
Sudip Dosanjh Sandia National Laboratories	ssdosan@sandia.gov
Daniel Hitchcock Office of Science	sally.mcpherson@science.doe.gov
William Spotz Department of Energy	wfspotz@ascr.doe.gov
Manojkumar Krishnan PNNL	manoj@pnl.gov
Andrew Uselton NERSC	acuselton@lbl.gov

Best Practices (1 of 4)

- Waterfall models do not work in HPC (application level)– Agile programming has become the defacto practice with continuous integration(except OS, tools, and file system). Test driven development has become standard. Tight development loop with:
  - Requirements
  - Development and documentation
  - Evaluation
  - Test
  - Deploy to early users and get feedback
  - Repeat above

Best Practices (2 of 4)

- Design for minimized maintenance and functional success
  - Work with hardware vendors for early testing at vendor’s and customer’s site of libraries, software, etc. API design collaboration
  - Test-driven software design and development
  - Thorough testing
    - Unit testing of code
    - Functional testing
    - System testing
    - Integration testing
    - All testing at scale (when possible)
  - Release criteria (synthetic workload tests)
  - Dedicated system time for tests at scale

Best Practices (3 of 4)

- Application-level testing of functionality during development and acceptance
  - Instrument codes to track usage
- PathForward – identification of gaps and direction of funding toward addressing technology gaps (e.g. Lustre file system resulted from this)

Best Practices (4 of 4)

- Tiger teams (cross functional multi-disciplinary teams and including vendors) –
  - During the life-cycle of the system to solve either focused or end-to-end problems and address issues
  - Working with vendor
- Issue tracking on all systems – using systems like TRAC and Jira
  - Includes hooks into underlying code repositories
  - E-mail developer directly and start an issue

Challenges (1 of 4)

- Agile software programming model
  - Software scope creep versus valuable features – how do we differentiate?
  - hard to put together a detailed schedule and budget for development
- Getting a holistic view to understand interactions between layers and repercussions of changes. Methodology for continually test to check functioning end-to-end. (e.g. library has its own build/test environment and each package has its own.
- Retiring software and systems – old versions and end of life reasons
  - Who is responsible for compatibility?
  - Process for roll-out of new versions
  - New versions of xxx – causing recompile of all apps and libraries

Challenges (2 of 4)

- Testing at scale and with end-to-end applications is important but keeping this test suite up-to-date is difficult
- Vendors don’t necessarily have the perspective of the users and more interaction during development is needed
- Managing the different executables and compilers – hard when everything has to be recompiled

Challenges (3 of 4)

- Funding does not have a holistic view - No explicit funding for migrating applications to new xxx
  – Dealing with new programming models, new architectures and scales, new libraries, new design paradigms,…
  – Hard to predict what will be coming – current practice is to study the range of expected architectures
  – Changes for future disruptive architectures often cause failures on existing architectures
  – What is the evolutionary path?

Challenges (4 of 4)

- Predicting when a new architecture will be deployed in machines – when should the software start migrating to the new architecture
- Getting more insight into architecture at the chip level to handle resource contention and related issues

New Technologies (1 of 2)

- System resilience – need a holistic approach to the system resilience as a whole (hardware and software)
- Methods for diagnosing problems at scale better
  - Ability to triage and debug
  - Visualization tools for debugging
  - Better summary data about how the system is behaving

New Technologies (2 of 2)

- Many core – programming models for dealing with this transition
- Simulators or emulators for systems at scale
  - Using tools to predict performance on new architectures – need better tools

Survey Questions (1 of 4)

- How valuable would a simulation/emulation capability for new architectures at scale be to you?
  - Scale of 1-4 with 4 being the most important
  - What level is important
    - Chip
    - System
    - System with applications

Survey Questions (2 of 4)

- Should a cross-facility committee be formed to identify technology gaps and sustainability issues to identify priorities for accelerated development and ideally impact funding (e.g. PathForward)?
  - Yes
  - No
  - No opinion

Survey Questions (3 of 4)

- What is your strategy for addressing the change to programming models for many-core?
  - 1) we are ignoring it
  - 2) we have formed a committee but no results yet
  - 3) the vendors and underlying libraries will handle this seamlessly
  - 4) We are preparing for it and are shovel ready

Survey Questions (4 of 4)

- Would you like to see improved diagnostics and what types of diagnostics would you like to have?
  - System level
  - Application level
  - Library level
  - Hardware level